SYSTEM AND METHOD FOR SPECIFYING APPLICATION SERVICES AND DISTRIBUTING THEM ACROSS MULTIPLE PROCESSORS USING XML

Field of the Invention

This invention relates to the field of distributed data processing systems, and, more specifically, to a system and method that uses XML to specify, validate and distribute functions over multiple processors in the distributed system.

Background of the Invention

Today's data networks are becoming increasingly complex to meet the growing needs of users. Sophisticated data networks collect data, store data, analyze data, present data to users, *etc.*, so that a user of this data may make informed decisions. Frequently, these tasks need to be carried out on more than one platform. Therefore, processing is distributed across many platforms (processors) in the network. In this context, a data client and the client's desired service platform may be one, two or several processors in the network.

XML provides a convenient mechanism for representing calls from a client application to a service application. It is simple to use, precise, flexible, portable and allows exchange of such requests between diverse applications, running on diverse platforms. An example of such an interface is the SOAP protocol (specified at: http://www.w3c.org/TR/2003/PR-soap-part0-20030507/). The SOAP specification essentially considers data packaged inside a request. The SOAP specification is mainly concerned with constraining the structure of the SOAP message.

However, SOAP cannot provide new functionality without changing the operational programs of client applications and service applications, especially if the applications are on distributed processors. Therefore, a problem in the art is that there is no simple method to specify new functionality that may potentially affect distributed data processing systems.

Summary of the Invention

This problem is solved and a technical advance is achieved in the art by a system and method that uses XML schemas to specify the set of allowed calls, to invoke multiple function calls, either hierarchically (*i.e.*, nested) or sequentially, and to specify the calls in such a way that they can be distributed across multiple processors.

In an exemplary embodiment of this invention, an XML schema mechanism defines the format of permitted requests and, importantly, validates the requests to ensure that they match the published signature of the functions provided by the service application. Additionally, the

schema mechanism validates the request to ensure that it matches the published signature of any further service application that it may need to employ. In the context of this specification, "signature" means the number and data type of the various parameters that must be passed to a function, along with its return type (*i.e.*, the meaning that is typically employed when dealing with traditional programming languages, such as C, C++ or Java). In addition, supplementary information added either within the schema itself or in supporting documents may be processed programmatically (*e.g.,* generating data input screens, performing performance simulations, generating test data, *etc.*).

A typical request may consist of a hierarchy of function calls, either nested within one another or executing in series. This relationship can be made implicit in the request, although the relationship may be supplemented by explicit details. The hierarchy of function calls may also contain additional content that indicates the points at which it may be beneficial to distribute the processing across multiple secondary service applications.

Thus, the exemplary embodiment of this invention uses an XML document both to represent the initial request, containing markup to indicate where such distribution may be performed, and to provide the basis for subsequent requests.

Brief Description of the Drawings

A more complete understanding of this invention may be obtained from a consideration of this specification taken in conjunction with the drawings, in which:

FIG. 1is a block diagram of a simple XML specification demonstrating aspects of this invention;

FIG. 2 is an example of a summing program;

FIG. 3 is an example of the summing program of FIG. 2 specified to be performed in parallel;

FIG.'s 4-10 comprise a sample XML schema instance definitions for the purpose of illustrating the exemplary embodiment of this invention; and

FIG.'s 11 - 13 are exemplary use of the schema of FIG.'s 4 - 10.

Detailed Description

As stated above, XML schemas provide a convenient mechanism for representing calls from a client application to a service application. An exemplary embodiment of this invention extends XML schemas to include nested operations that may either be executed locally on the

receiving service application or may be passed on to subsequent service applications (*i.e.*, the current service application becomes the client for these secondary service applications).

The execution of both the caller and the service side of this mechanism is most advantageously performed in real-time, such that the client would construct the message as needed and pass this on to the service processor. The service processor would interrogate the message as it arrived and determine the appropriate response based on its content.

If a real-time approach is not possible, due to technical or other limitations, then an alternative approach is to use the schema (with possible supplementary details held either within the schema or separately) as the basis for generating the necessary programs to perform these operations. The generated programs could include any necessary steps required to integrate the software with the target runtime environment, *e.g.,* to interface with any grid management software that may be employed. A grid computing framework is generally defined as a series of two or more processing units that can be used to support distributed processing, whereby a single unit of work may be broken up to be performed in parallel by each of the processing units. Thus, the overall end-to-end processing time is reduced. Grid management software is used to support this operation, for example, to optimize the distribution process, to support monitoring of the processing and to allow recovery in the event of errors. This alternative embodiment avoids the need for programmers to write all of the necessary code. It also simplifies the task of migrating to a different environment using a different set of interfaces, because this would merely involve modifying the code generator and then re-generating all of the necessary application code.

XML is also a hierarchy of named items, as is known in the art. The mapping onto a nested series of operations and their parameters according to the exemplary embodiment of this invention will be clear to one skilled in the art after studying this specification.

In this exemplary embodiment of this invention, the basic pattern encountered is a simple Composite:

- An operation has a name and takes a number of arguments...
- The arguments may either be simple value operands...
- Or child operations

Theoretically, this pattern can be nested arbitrarily. The limitation of nesting will be dependant upon the specific implementation. This pattern is also common in expression evaluation such as:

Result = a + (b * func1(c – func2(d, e+4)))

The above equation illustrates that an expression may generally be viewed as consisting of two operands and an operator, which is typically placed between them, in which each operand may itself be an expression.

Because the input can be presented as XML, XML Schema may be utilized to define valid input formats. For example, a rather "open" specification might look like FIG. 1 (note that the range of data types is deliberately limited here for clarity). XML Editors (*e.g.,* XMLSpy, http://www.xmlspy.com) can use this to prevent the construction of invalid data, *i.e.,* manually edited input (*e.g.*, for test data). It can also be used to automatically validate the data at either the client or server side of the interface assuming the use of a validating parser (*e.g.,* Xerces, http://xml.apache.org/xerces2-j/index.html). Schemas are generally accessed using URI embedded in the document to be validated. Such validation enables these to be maintained centrally (*e.g.,* accessed over an intranet).

As stated above, it is possible to generate client-side code from input data. However, the fact that this is possible implies that the intermediate code may be cut out completely. Rather than generating code, calls are made directly from within the "code generator" itself. Such direct calls provide a generic evaluation engine, which is capable of passing data to any function.

Schemas are themselves represented as XML documents, that is, there is a schema that defines valid schemas. These schemas can be read and processed in much the same way as any other XML file. For example:

- To generate data input screens
- To generate test data
- Most any other use

Server-side processing typically consists of extracting the data from the maps, validating it *(e.g.,* testing the data type) and passing this through to, for example, "C" code. The transport from this point onwards is usually either simple values or more complex structures. According to this exemplary embodiment, it should be possible to code-generate this code too. However, the data/specification will need supplementary details, for example, the names of the structures and fields into which the data is to be put. In the examples shown in FIG.'s 2 and 3, this is done using the standard "xs:appinfo" mark up facility, although other mechanisms should be evident

to one skilled in the art after studying this specification. This could simplify the migration to the new approach and thus lead to shorter development times.

In the example of FIG. 2, code to sum the result of a large number of calculations is shown. The "natural" way of executing this would be to perform each calculation in series. However, if the input is modified in accordance with the code of FIG. 3, then code can be generated that performs the operation in parallel. This could be performed on either the client or the server side of the interface.

Further, the above could greatly reduce the overhead of migrating code to a grid computing framework by generating the wrapper code that is required for deployment. Further, this could be modified to use any alternative Grid that might be used in the future. For example, envisage a farm of data-driven evaluation engines, which parse their input and execute on-the-fly. This may possibly break up their input and pass it on to other evaluation engines. The forwarded request would essentially consist of a fragment of XML from the original request (with a small amount of "wrapping"). (As an aside, further annotation, with execution timings, could be used to simulate scenarios and predict timings, *e.g.,* to allow worst-case processing times to be estimated).

Further extensions could allow chaining of results (plus many other possible features). For example, the above-described approach does not care where "splitting" occurs (*i.e.* client-side or server side). It retains the flexibility and loose coupling of the proposed new approach, while, significantly, adding a degree of formalism where appropriate. Services (*i.e.,* operation providers) could be discovered on the fly, e.g. using UDDI (http://www.uddi/org), for example, just "throw" a request document at a generic calculation service and then wait for the result.

FIG.'s 4 – 10 comprise an annotated XML schema. Definitions of elements are shown in these drawings as "documentation."

FIG.'s 11 - 13 illustrate an XML schema that shows a number of examples. These examples make use of the schema of FIG.'s 4 -10, above. These examples can be validated against this schema. Note that the substitution group facility allows nesting of function calls arbitrarily, while ensuring that type-correctness is maintained (*e.g.,* an Integer value or function cannot be passed to the Add function).

It is to be understood that the above-described embodiment is merely illustrative of the present invention and that many variations of the above-described embodiment can be devised

by one skilled in the art without departing from the scope of the invention. It is therefore intended that such variations be included within the scope of the following claims and their equivalents.